Petri Net Based Hypertext: Document Structure with Browsing Semantics

P. David Stotts
Richard Furuta*

Department of Computer Science and
Institute for Advanced Computer Studies
University of Maryland
College Park, MD 20742

Abstract

We present a formal definition of the Trellis model of hypertext and describe an authoring and browsing prototype called αTrellis that is based on the model. The Trellis model not only represents the relationships that tie individual pieces of information together into a document (i.e., the adjacencies) but specifies the browsing semantics to be associated with the hypertext as well (i.e., the manner in which the information is to be visited and presented). The model is based on Petri nets, and is a generalization of existing directed graph based forms of hypertext. The Petri net basis permits more powerful specification of what is to be displayed when a hypertext is browsed and permits application of previously-developed Petri net analysis techniques to verify properties of the hypertext. A number of useful hypertext constructs, easily described in the Trellis model, are presented. These include the synchronization of simultaneous traversals of separate paths through a hypertext, the incorporation of access controls into a hypertext (i.e., specifying nodes that can be proven to be accessible only to certain classes of browsers), and construction of multiple specialized (tailored) versions from a single hypertext.

CR categories: I.7.m (hypertext), F.1.1 (Petri nets), H.3.4

General terms: hypertext, Petri nets, browsing semantics, formal model, synchronization, access control, versions

Additional Key Words and Phrases: Trellis model of hypertext

1 Introduction

Hypertext is a structuring mechanism for information, one that is particularly well-suited for use on an interactive computer. Hypertext is not a new idea. Bush [3] is credited with the first proposal for such a system, which he called the "memex," in 1945. Brown University's HES [5] and Engelbart's NLS [13, 14] are early implementations of computer-based hypertext systems, dating from the late 1960's. Only in recent years have hypertext systems become widely available. Commercially-available systems such as Apple's HyperCard and Shneiderman's Hyperties [25, 21] may be purchased for use on personal computers. Research systems such as Xerox's NoteCards [17] show the wide range of components that are useful in a hypertext and the different presentations that help make it more understandable.[1]

Traditionally, a hypertext is composed of information fragments (text, graphics, sound, video, etc.) and tangible relationships among these fragments. One way to represent such a document is as a directed graph. Each information fragment is associated with a node in the graph, and the directed arcs represent the

*Supported in part by a grant from the National Science Foundation, CCR–8810312.

[1] Two recent surveys of the hypertext field are [8] and [2]. In addition, [30] presents a perspective on the history and development of the area.

relationships among nodes. In this interpretation, the arc represents a potential traversal from the source node of the arc to the destination node. A person *browsing* a hypertext traverses the graph and views (or hears) the information fragments as he visits nodes.

Some researchers are beginning to realize the need for a more formal substructure for the highly implementation-defined field of hypertext; for example [12] and [16]. Projects are beginning to appear that attempt to construct a more complete and descriptive mathematical basis for documents than directed graphs and annotations can provide.

The work described in this report is based on one such mathematical framework—that provided by a Petri net. The Trellis model essentially provides a unifying formalism for describing and reasoning about many of the features of existing hypertext systems. The model also naturally expresses and manages concurrent browsing paths, an area of hypertext that has not been fully explored. Another advantage of this approach is that a Petri net is an incremental change to the commonly understood directed graph formalism, not a wholesale replacement of that representation, and results developed for directed graph based hypertext are easily adapted to a Petri net formalism. Unlike a directed graph alone, though, a Petri net also permits the specification of a hypertext's *browsing semantics*, that is, the dynamic properties of a reader's experience when browsing a document. This is due to the dual nature of Petri nets: they are easily and naturally represented as bipartite directed graphs, but they are automata as well and have inherently parallel execution semantics. As automata they also have formal language properties and can be viewed as language generators and recognizers. These different but interchangeable aspects of nets provide analytical leverage for the solution of several interesting access control and version problems in hypertext.

Petri net formalism has long been recognized as as effective tool for describing and analyzing control flow among concurrent activities. Though we know of no other use of Petri nets in describing hypertexts, Petri nets have been used to model user interactions and interfaces. For example, Zisman [34] describes a model for activity coordination in office automation environments in which Petri nets are used to specify the possible interactions among a set of active agents. The agents themselves are represented as production systems that specify computational behavior. In another project, van Biljon [29] has used Petri nets to specify user interfaces, calling them man-machine dialogues. For illustration he models a simple operating system shell. These practical uses of Petri net theory are similar in goals and approach to the Trellis model described herein, but our work is cast directly into the hypertext domain and addresses some of the unique problems found there.

In the following section of this paper we present a brief review of Petri nets and then we present our formal hypertext model. Following the presentation of the model, Section 3 describes αTrellis, a prototype Petri net based hypertext system. Section 4 then presents the solution to some specific hypertext problems as an illustration of the analytic power of our formalism. A discussion of useful extensions and further implications of the Trellis model, in Section 5, completes the report.

2 The Trellis model of hypertext

The formal model of hypertext that we propose is based primarily on a Petri net representation of a document's structure. We take advantage of the fact that Petri nets not only capture the descriptive power of directed graphs, known to be a useful abstraction in hypertext systems, but provide as well a mathematically precise abstract machine for control and analysis of hypertext "execution," or "browsing."

For completeness of our discussion, and to motivate the descriptive power of our approach, we first provide a short set of definitions for the Petri nets we employ. Following them, we give the definitions for our hypertext model. Readers familiar with basic Petri net theory can skip directly to Section 2.2 without loss of understanding.

2.1 Basic Petri net theory

To provide a complete presentation of the Trellis hypertext model, we review first the basic points from Petri net theory that are used in the later definitions. Readers who desire a more thorough exposition of net

theory can consult the texts by Peterson [23] and Reisig [24]. The notation used here is taken from Reisig.

Definition 1 Petri net structure

A *Petri net structure* is a triple, $N =< S,\ T,\ F >$ in which

$S = \{s_1, \ldots,\ s_n\}$ is a finite set of *places* with $n \geq 0$,

$T = \{t_1, \ldots,\ t_m\}$ is a finite set of *transitions* with $m \geq 0$, and $S \cap T = \emptyset$,

$F \subseteq (S \times T) \cup (T \times S)$ is the *flow relation*, a mapping representing arcs between places and transitions.

The arcs represented by F prescribe *pre* and *post* relations for places and transitions. The set of places that are incident on a transition t is termed the *preset* of t and is denoted by

$$\bullet t = \{s \mid (s,\ t) \in F\}.$$

The set of places that follow a transition t is termed the *postset* of t and is denoted by

$$t\bullet = \{s \mid (t,\ s) \in F\}.$$

The preset and postset for a place s are defined similarly as the sets of transitions incident on s and following s respectively.

In our model, we have simplified the notation often used for Petri nets by assuming that the weight on each arc is 1, and that the token capacity of each place is unbounded.

Definition 2 Marking

A *marking* M of a Petri net structure $N =< S,\ T,\ F >$ is a function

$$M : S \rightarrow \{0,\ 1,\ \ldots\} \cup \{\omega\},$$

mapping each place in the net into either a nonnegative integer or the symbol ω. A marking is normally written as a vector $(m_1, m_2, \ldots, m_n)$ in which $m_i = M(s_i)$.

Each integer in a marking indicates the number of *tokens* residing in the corresponding place. The symbol ω represents an arbitrarily large number of tokens, and is included for consistency with existing net theory. It may appear in Petri nets representing an infinite number of states.

An execution of a Petri net consists of a sequence of markings, beginning with the initial marking M_0 and ending in some final marking M_f (which can be defined in several different ways; see [23]). To go from the current state M to some next state M', any transition t that is enabled under M is chosen and fired. For a transition t to be enabled under M, it must be the case that

$$\forall s \in \bullet t : M(s) \geq 1.$$

Firing t consists of removing one token from each place in $\bullet t$ and adding one token to each place in $t\bullet$. The new state M' is then the tuple of integers showing the number of tokens in each place after the firing.

For example, consider the marked Petri net shown in Figure 1. Places are drawn as circles, transitions as bars, and tokens as dots in places. The initial marking, shown in part (a), is $M_0 = (111000)$. The enabled transitions under M_0 are t_1 and t_2. If t_1 were chosen to fire, the resulting next state would be $M_1 = (001111)$, as shown in part (b). If instead from the initial state transition t_2 were fired, the resulting state (100000) would be terminal since no further transitions would be enabled. Note that the number of tokens in s_4 is increased by 1 every time that t_1 is fired; hence s_4 acts as a counter.

Being a finite automaton, a Petri net has a dual mathematical nature. It can be viewed as generating (or representing) a formal language, and it can also be viewed as an abstract machine. From the formal language viewpoint, a Petri net describes a set of strings of symbols in which each symbol represents a transition in the net. From the automaton viewpoint, a Petri net is a state transition system in which the number of tokens in each place collectively constitutes the state of the abstract machine. Both views are of value to us when interpreting a Petri net in terms of a hypertext.

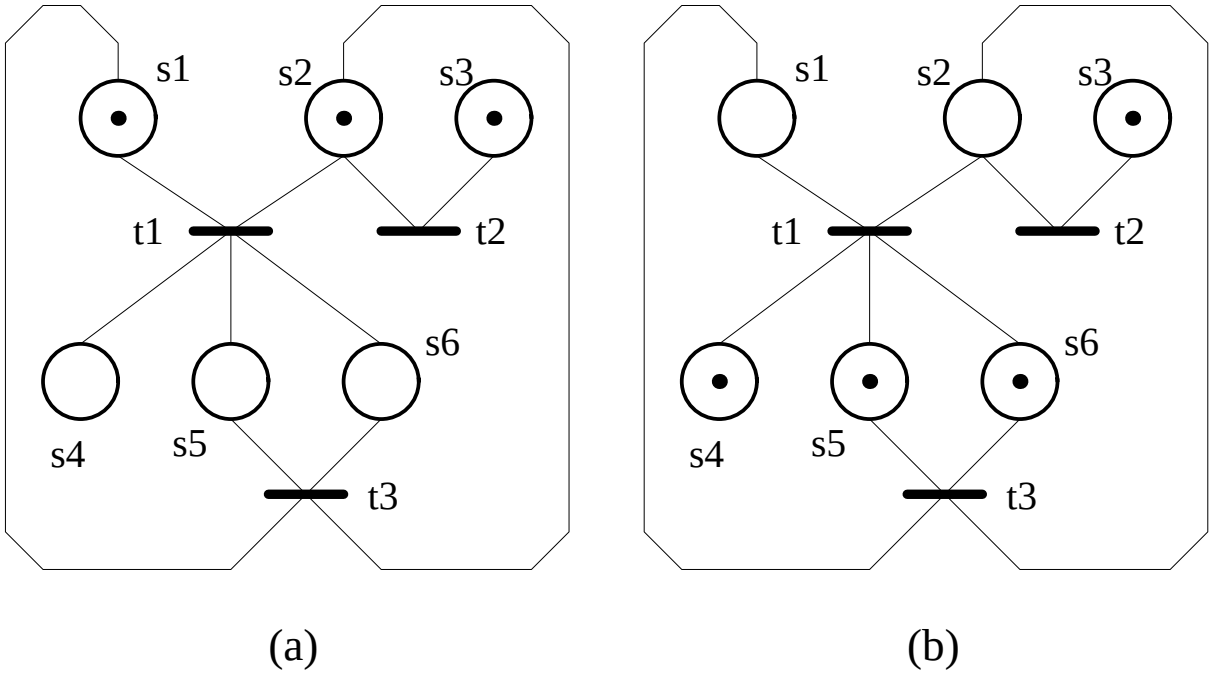


Figure 1: Petri net before and after transition firing.

2.2 Formal definitions of Trellis

The Trellis model of hypertext uses Petri net structure and execution semantics to specify both the linked form and the browsing semantics of a hypertext. This logical structure then is interpreted through a layer of indirection to arrive at a displayed form for reader consumption. Hypertext content and linked structure are effectively separated by the Trellis model.

Definition 3 Hypertext
A *hypertext* H is a sextuple $H =< N,\ C,\ W,\ B,\ P_l,\ P_d >$ in which

$N =< S,\ T,\ F >$ is a Petri net structure,
C is a set of document *contents*,
W is a set of *windows*,
B is a set of *buttons*,
P_l is a logical projection for the document,
P_d is a display projection for the document.

A hypertext consists of a Petri net representing the document's linked structure, several sets of human-consumable components (*contents*, *windows*, and *buttons*), and two collections of mappings, termed *projections*, between the Petri net, the human-consumables, and the display mechanisms. A Petri net is a bipartite directed graph, and as such, is as capable of representing a linked hypertext as existing models using directed graphs alone.[2] A window from set W is a logically distinct locus of information.[3] A button from set B is an

[2] Any directed graph has a simple representation as a Petri net; the bipartite nature of a Petri net is important for the execution semantics and does not restrict its structural modeling power. See Section 5.

[3] The locus of information represented by a "window" is not necessarily visible. Indeed, a logical window may require other devices, such as an audio generator, for presentation of its information. Note that the model does not require that a logical window be associated with each net place (see the definition of Logical Projection, below). This is useful when a place is mapped to a computation that does not create presentable information, e.g., an action node in NoteCards.

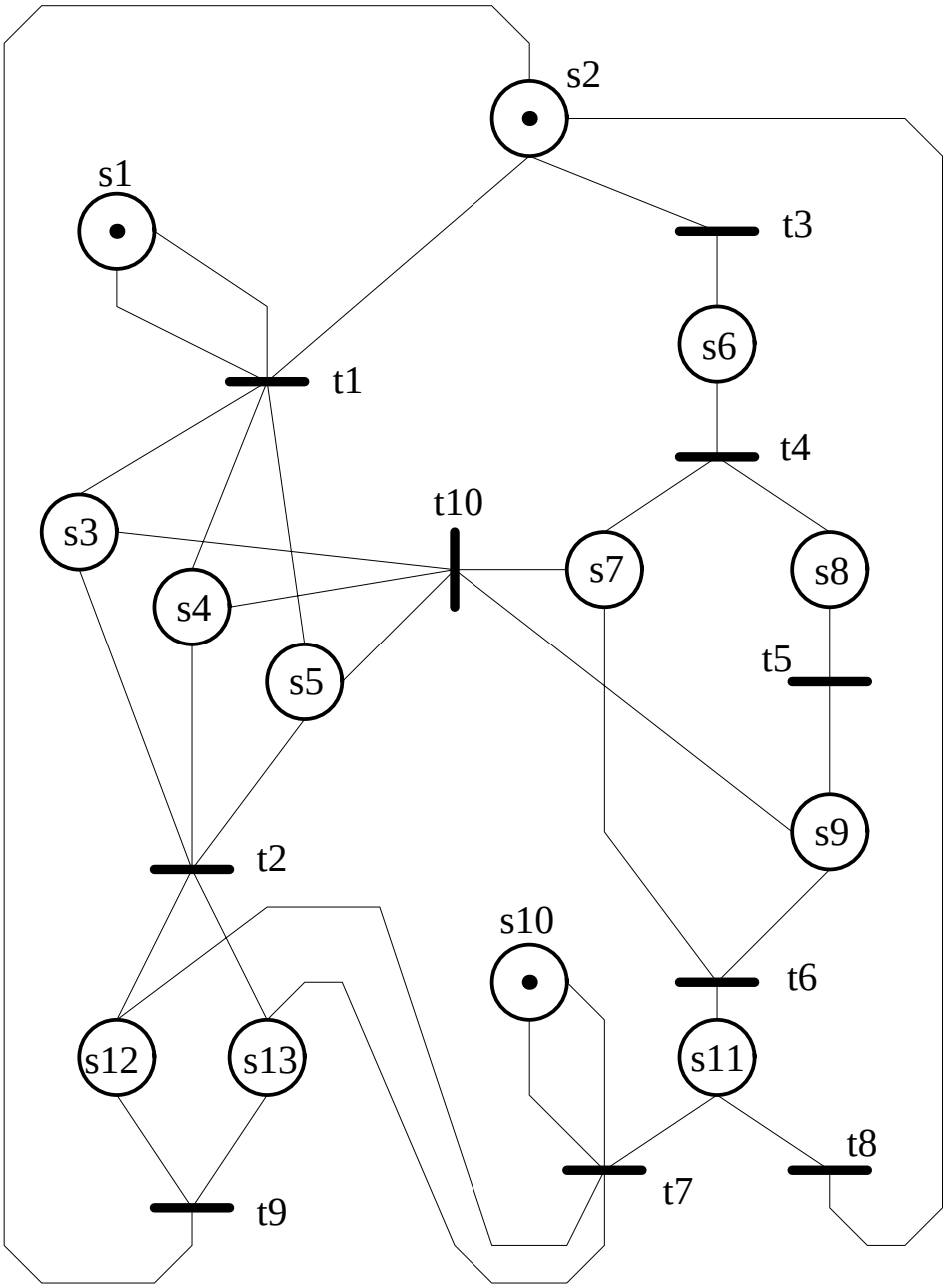


Figure 2: Example Petri net hypertext structure

action that causes the current display to change in a specified way. A content element from the set C can be many things: text, graphics, tables, bit maps, executable code, audio information, or, most importantly, another hypertext. Figure 2 shows an example of a Petri net structure for a hypertext. It will be used as the basis for further examples.

Definition 4 Logical Projection

Given the Petri net structure $N =< S,\ T,\ F >$ in a hypertext H, the *logical projection* of H is a triple $P_l = < C_l,\ W_l,\ B_l >$ in which

$C_l : S \rightarrow C \cup \{\nu\}$
$W_l : S \rightarrow W \cup \{\nu\}$
$B_l : T \rightarrow B \cup \{\nu\}$

A logical projection provides mappings from components of a Petri net to the human-consumable portions of a hypertext as mentioned above. The ν in the definition is a special item denoting a null value. The function C_l associates a *content* element with each place in the Petri net N. The function W_l associates a *logical window* with each place in N as well. The function B_l associates a *logical button* with each transition in N. A transition is fired by selecting its associated logical button. Buttons can thus be thought of as the mechanism that couples hypertext browsing with Petri net execution. Together, these mappings provide an informational form and a logical layout for the browsing structure of the Petri net.

As mentioned above, content elements may be more than simple information. A Petri net place may have as its contents another hypertext, creating a natural hierarchy in the model. The uses for hierarchical structure in a hypertext are numerous. One obvious advantage is that each Petri net is independently analyzable. Many analysis algorithms for Petri nets are exponential in the size of the net, so a collection of smaller nets is preferable to one large net. Another advantage is that with careful use of hierarchy an author can organize a hypertext more effectively than with a flat structure. For instance, hierarchy can provides a form of abstraction. Browsing semantics can be designed so that the initial display of a lower level hypertext shows an overview of its contents. The detailed text can then be browsed for more information, or it can be skipped by moving on at the upper level.

The *display projection* P_d is simply a collection of mappings that associate the logical buttons and windows of a hypertext with physical screen representations and locations. For example, given a text segment displayed in a physical window, one way to represent the logical buttons is to map them to a menu of mouse-selectable items located next to the text window; this is the approach in the current αTrellis system, explained in Section 3 below. Another way is to map them onto words actually in the text window. Either mechanism (or others) can be used without altering the logical hypertext structure simply by appropriately specifying the display projection. Since our emphasis here is on logical representation and analysis, we will not present this aspect of the model in any detail.

There are, in summary, two important levels of indirection in this model. The first is the separation of content from structure. By appropriate selection of the logical mappings P_l one structure can serve to represent several document versions (see Section 4.5). The second level is the display projection P_d, which allows a set of logical entities to be presented in different forms for consumption. For example, logical buttons that exist but are not selectable (due to the Petri net marking) can either be shown on a screen "grayed out" (as in the well known Apple Macintosh menu format), or they can simply not be displayed at all. This second approach has implications for document security: if information is not attainable, its existence is not even admitted (see Section 4.4).

Definition 5 Marked Hypertext
A *marked hypertext* is a pair $H_M =< H,\ M >$ in which

$H =< N,\ C,\ W,\ B,\ P_l,\ P_d >$ is a hypertext,
M is a marking for the Petri net N in H.

A marked hypertext can be thought of as representing the state of a hypertext during browsing. It is a characterization of the set of possible paths through a hypertext from a given point. A special case of marked hypertext, termed the *initial state*, is $H_{M_0} =< H,\ M_0 >$ where M_0 is an initial marking for the Petri net in H. Different browsing patterns can be enforced on a single hypertext simply by choosing appropriate initial states, as demonstrated in the section below on access control restrictions. When a hypertext is first viewed, the node contents displayed correspond to the places that contain tokens in the initial net marking M_0; that is,

$$\{C_l(s) \mid s \in S \ and \ M_0(s) > 0\}$$

is the set of elements displayed.

The execution semantics of a Petri net provides the model of browsing a marked hypertext. A token in a place s indicates that the contents of the place $C_l(s)$ are displayed for viewing (or editing, or some other interaction). When a token moves into an empty place, the associated content element is mapped to the display device; likewise, when all tokens are removed from a place, leaving it empty, its contents are removed from the display. Tokens move through the net as transitions are fired. This is accomplished by selecting logical buttons in the display. When a transition t is enabled in the Petri net, the logical button $B_l(t)$ is mapped to some area of the screen where it may be selected.[4] Browsing begins by starting execution of the Petri net in M_0. Browsing may terminate, depending on the structure of the hypertext, or it may cycle without end. If a state M of the Petri net is ever reached in which no transitions are enabled, then browsing ends, since no buttons are selectable at that point.

Hierarchy affects execution is the following way. When a token arrives at a place s having a marked hypertext as its value under the content mapping C_l, the content elements corresponding to the places initially marked in the lower-level net $C_l(s)$ are displayed. Browsing in the lower-level net continues concurrently with the remainder of the marked locations in the higher-level net. The transitions leaving place s at the higher level remain visible for selection as long as they are enabled (this reflects the idea that as long as the lower-level net is being browsed, a token is sitting in the higher-level place representing it). Selection of one of the buttons on arcs out of s causes immediate termination of the lower-level browsing session. Otherwise, the lower-level net may be browsed until no transitions in it are enabled. At this point, the reader has only the higher-level transitions to select leaving s, and browsing will continue at the higher level.

To summarize, a marked hypertext combines graphical structure with Petri net execution semantics to encapsulate all possible paths that a reader may follow through a document. During browsing, the current marking M of the Petri net determines which hypertext elements are viewable. The transitions enabled under M determine which buttons are visible, and hence selectable, in which windows. Selection of a button by the reader fires one of the enabled transitions, thereby generating a new state M' from M and causing the display to change correspondingly. Browsing formally terminates when no buttons are selectable.

3 The αTrellis hypertext system

We have constructed a prototype hypertext browsing and authoring environment called αTrellis to experiment with the Petri-net-based Trellis hypertext model. It presents a multi-path browsing environment in which many different elements may be viewed at a given point in time. The current version of αTrellis operates on Sun-3 workstations under the SunView window package. αTrellis is intended as an experimental platform and a proof-of-principle vehicle. As such, we have paid more attention to implementing the Petri net document representation with browsing semantics, and have not implemented sophisticated window placement strategies.

αTrellis allows both construction and viewing of a Petri net hypertext. Physically, the αTrellis screen is divided into two main parts. On the right side is a Petri net editor and simulator, derived from Molloy's SPAN tool [22]. The left side of the screen is the hypertext browser, subdivided into four windows, each containing a text panel and a button panel. Using the net editor, an author builds a Petri net structure and then, using tag strings, specifies the mappings of places to browsable text elements and transitions to buttons. The tag on a place is the name of a Unix file containing the associated text. The tag on a transition is the name displayed in a button panel when that transition is enabled to fire. During browsing, when a token resides in a place, the text in the file associated with that place is displayed in one of the four text panels. The place name appears in the upper left hand corner of that window. Any enabled transitions in the postset of the place have their button names displayed in the button menu to the left of the text. Button names do not appear in any browser window until the associated transitions are enabled to fire.

Both the author and the reader can refer to the visual representation of the Petri net to get a graphical

[4] The choice of how to do this mapping is a design decision and is the function of the display mapping. The particular mapping chosen for the αTrellis prototype is to show the button in the window of *each* place in the preset of t. This is particularly useful in, say, a distributed implementation where the various windows may not appear on the same screen.

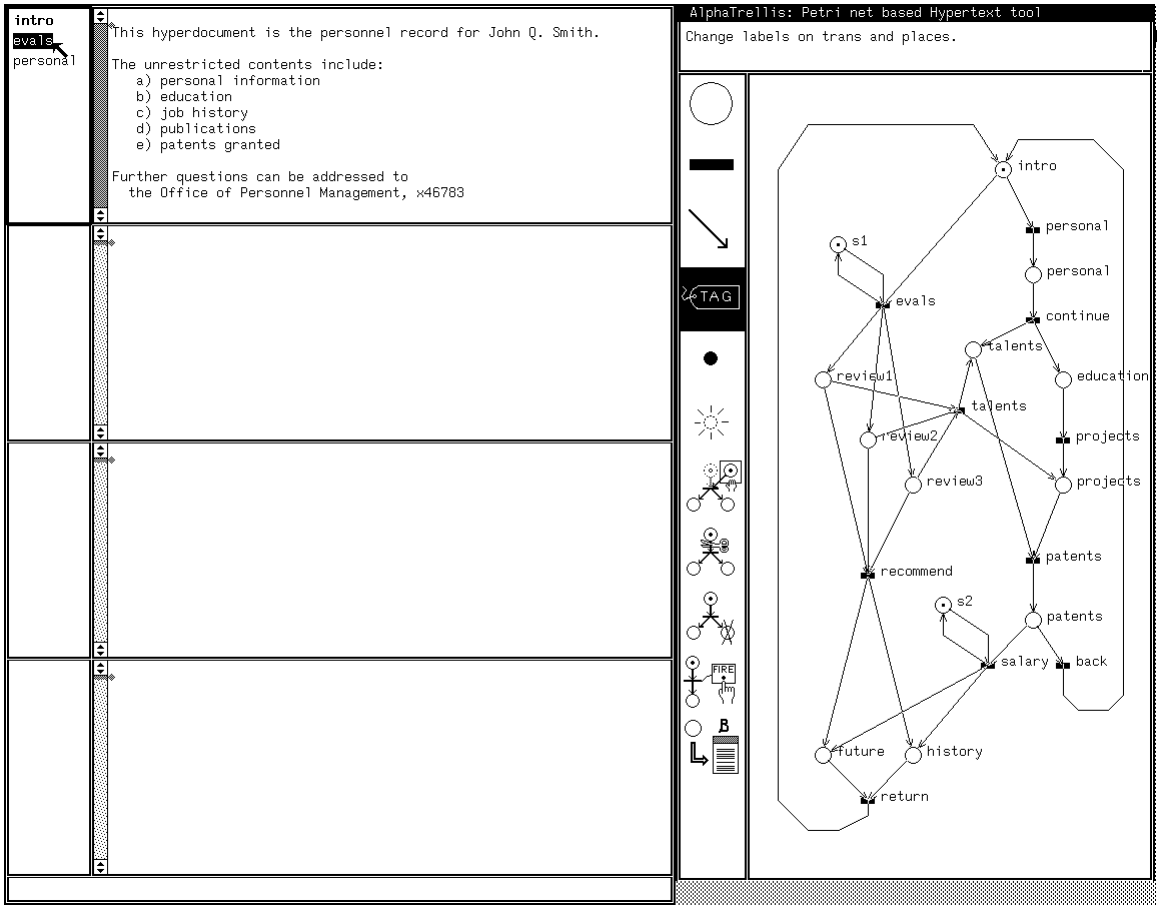


Figure 3: Screen from the αTrellis system.

image of his location(s) in the document. Only an author is allowed to alter a document structure or directly place tokens in the net. Reading a hypertext is accomplished by simply executing the Petri net from its initial marking. Execution is controlled either (as is common in hypertext systems) by mouse selection of displayed buttons in the browser windows, or by direct firing of enabled transitions from the net editor.

Figure 3 shows a initial screen from αTrellis, using the example shown earlier in Figure 2. This simple hypertext is a fictitious personnel record. Among other things, it contains salary, education, job history, and performance evaluation information for an employee. The Petri net structure of the hypertext is directly illustrated on the right side. The content mapping C_l is implicitly represented by the file names tagging the net places. The logical button mapping B_l is represented similarly by the tags on the transitions. The logical window mapping for this example simply associates a different logical window with each net place. The display projection implemented in αTrellis specifies that a logical window will be displayed in the first open panel on the left. If none are open, the extra logical windows are not displayed until some browsing action causes space to open up. The number of logical windows waiting for open space is noted at the bottom of the browser. In Figure 3, the button "evals" is being selected, causing the display shown in Figure 4 to appear. Notice that now there are three concurrently displayed elements. The net structure shows two different transitions leaving these places, and the buttons for these transitions are displayed next to each text panel. Here, button "recommend" in panel "review3" is being selected. Figure 5 shows the resulting screen display. All three previous windows have been removed together, and two more have been displayed as dictated by the net structure. Notice in these figures the token movement through the Petri net, corresponding to transitions fired by button selection.

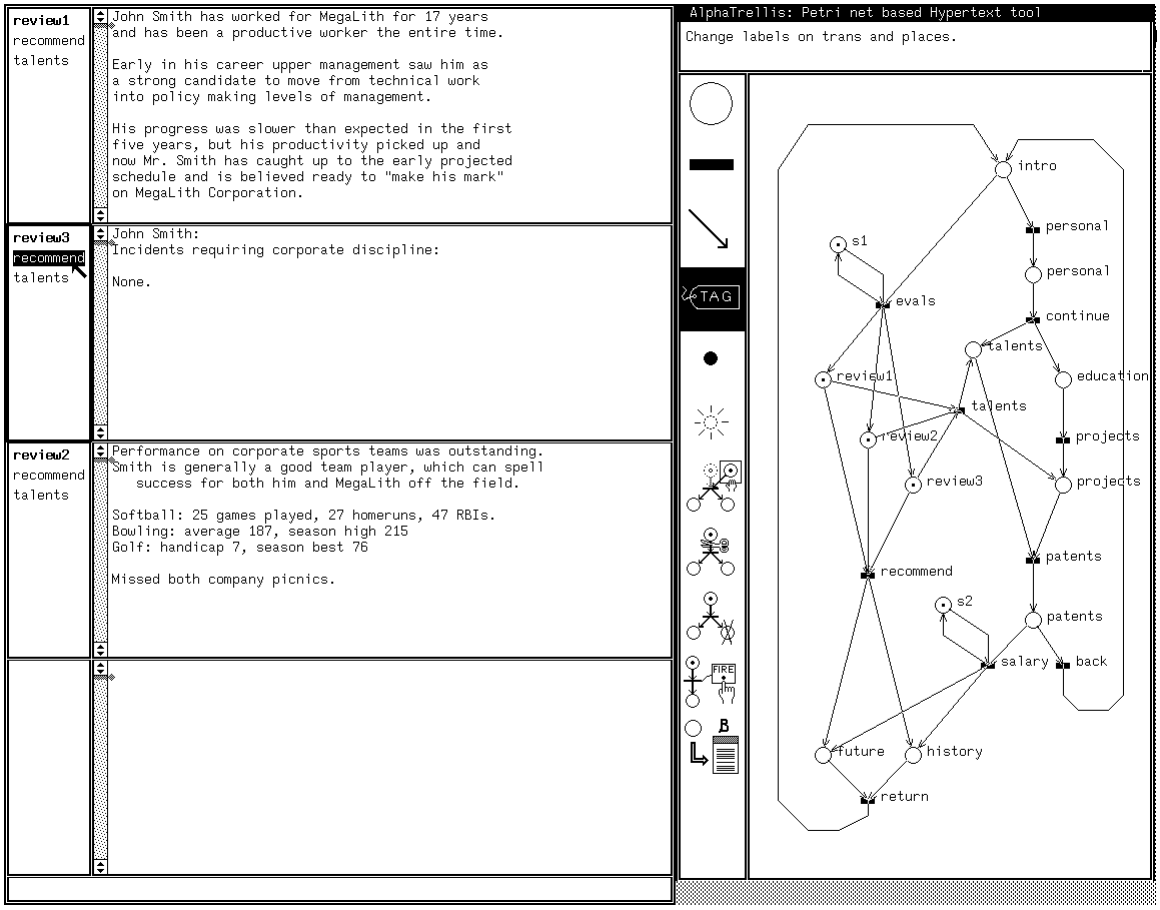


Figure 4: Browsing concurrent elements in αTrellis.

4 Solutions using the model

The Petri net basis for hypertext in the Trellis model allows the solution of several problems using state-space analysis techniques. In this section we discuss the following:

- *Display parameters:* characteristics of the display, such as the maximum number of simultaneously needed windows, can be determined for planning screen layouts.
- *Concurrent browsing paths and synchronization:* a Petri net based hypertext structure can naturally indicate that several content elements are to be viewed at the same time. More generally, an author can easily specify the creation and deletion of multiple concurrent browsing paths. These can be independent, or they can be related in various ways. An author can also specify that multiple paths be synchronized at certain places during browsing so that concurrently displayed information elements do not become unrelated to each other.
- *Node reachability and unreachability:* it is possible to verify that all nodes in a hypertext can be reached via some path; more importantly, verification is possible that certain nodes *cannot* be reached from particular initial markings, giving the basis for *access control* and for *tailored versions*.

Many of these solutions are derived from the *reachability graph*, a structure representing the various token distributions a Petri net may encounter during its execution. A marking can be thought of as the state of a Petri net automaton. The set of all markings a net can possibly attain during execution is called

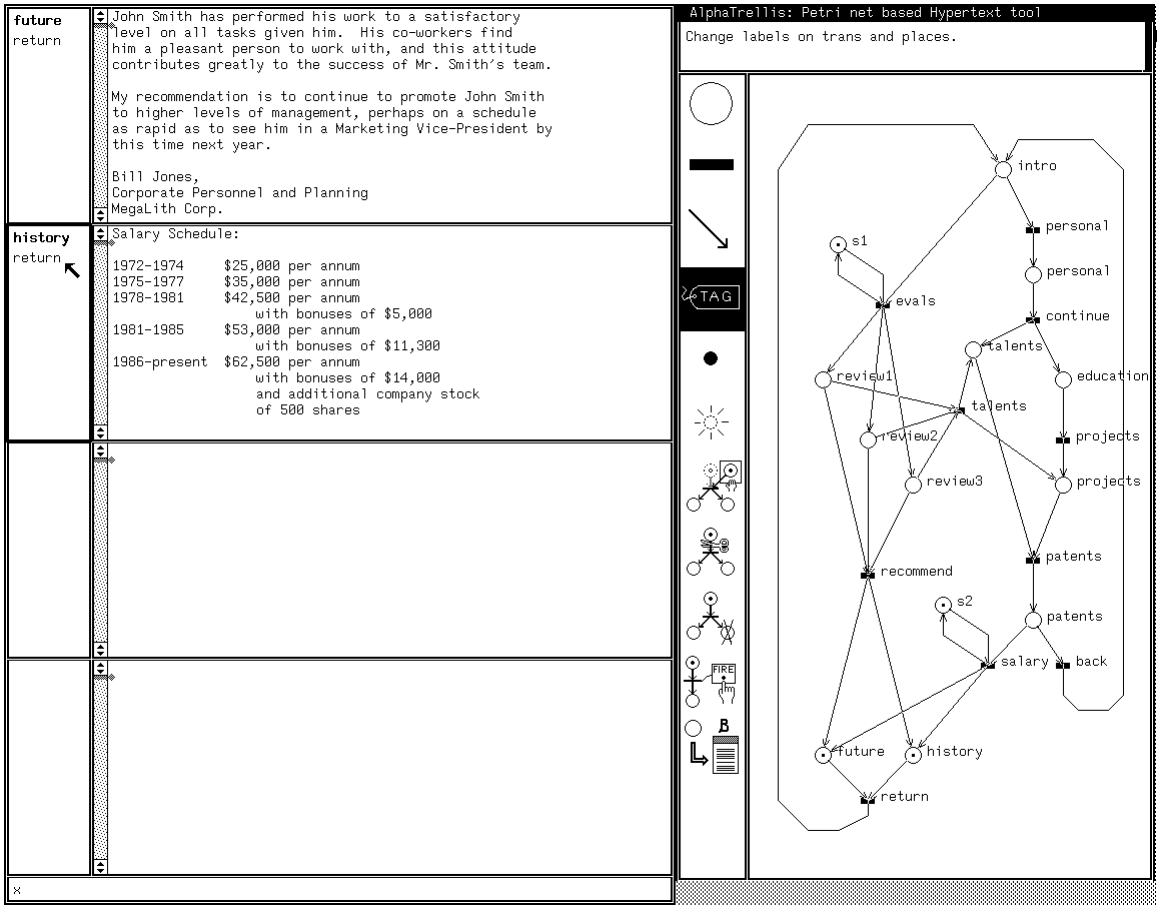


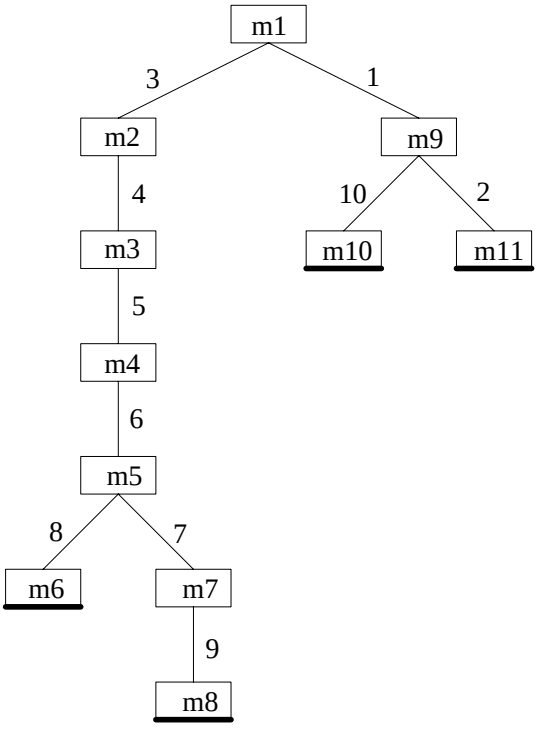
Figure 5: Synchronization of parallel paths in αTrellis.

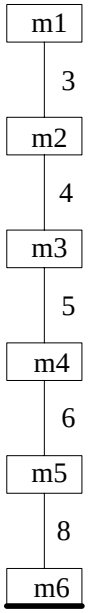
its *reachability set*. For a general Petri net, this set can be infinite, though for a Petri net representing a reasonable hypertext it will be finite. In his book, Peterson summarizes a technique for representing this possibly infinite set as a tree that is guaranteed to be of finite size for any Petri net [23, page 91, ff.]. The technique involves collapsing some infinite subsets of states into single meta-states.

The left side of Figure 6 shows the reachability graph for the Petri net in Figure 2. Each node in the tree structure is a net state; the state number is indicated in the box, and the marking for each state is listed in the table below the graph. Each arc leaving a node is labeled with the number of the transition that must fire to create the marking at the end of the arc. A node with a heavy line at its base is a duplicate of one found elsewhere in the graph. Note that the reachability graph is highly dependent on the initial marking. The right side of Figure 6, for example, shows the reachability graph for the same net structure as the one on the left, but with an initial marking $M_0 = (0100000000000)$.

4.1 Display complexity

Using the reachability graph for the Petri net in a hypertext, we can determine some parameters of the physical presentation that cannot be determined from a directed graph alone for many forms of hypertext browsing. One such parameter is the maximum number of windows that will be required for any reading of the hypertext. Since the model associates a content $C_l(s)$ with each place s in the Petri net, if the browser displays each concurrently viewed element in a separate window, then the number of marked places in a Petri net state is the number of windows required to be concurrently displayed for that state. We can then scan the reachability graph node by node and find the maximum number of marked places over all states. For





m1 : (1100000001000) is interior
m2 : (1000010001000) is interior
m3 : (1000001101000) is interior
m4 : (1000001011000) is interior
m5 : (1000000001100) is interior
m6 : (1100000001000) duplicates m1
m7 : (1000000001011) is interior
m8 : (1100000001000) duplicates m1
m9 : (1011100001000) is interior
m10: (1000001011000) duplicates m4
m11: (1000000001011) duplicates m7

(a)

m1 : (0100000000000) is interior
m2 : (0000010000000) is interior
m3 : (0000001100000) is interior
m4 : (0000001010000) is interior
m5 : (0000000000100) is interior
m6 : (0100000000000) duplicates m1

(b)

Figure 6: Reachability graphs for Petri net in Figure 2

the reachability graph shown in Figure 6, this number is 5. This information can be employed in a number of ways; for example, it can aid the determination of a reasonable layout for a display mechanism that, say, tiles a screen with windows.

The analysis presumes that concurrently viewed elements are specified by the author rather than the reader. This means that in a single browsing session, a reader is not allowed to simultaneously initiate separate traversals of the various alternative paths leaving a node. Systems allowing that form of browsing certainly exist. Rather than treating such behavior as a single session with arbitrary potential concurrency

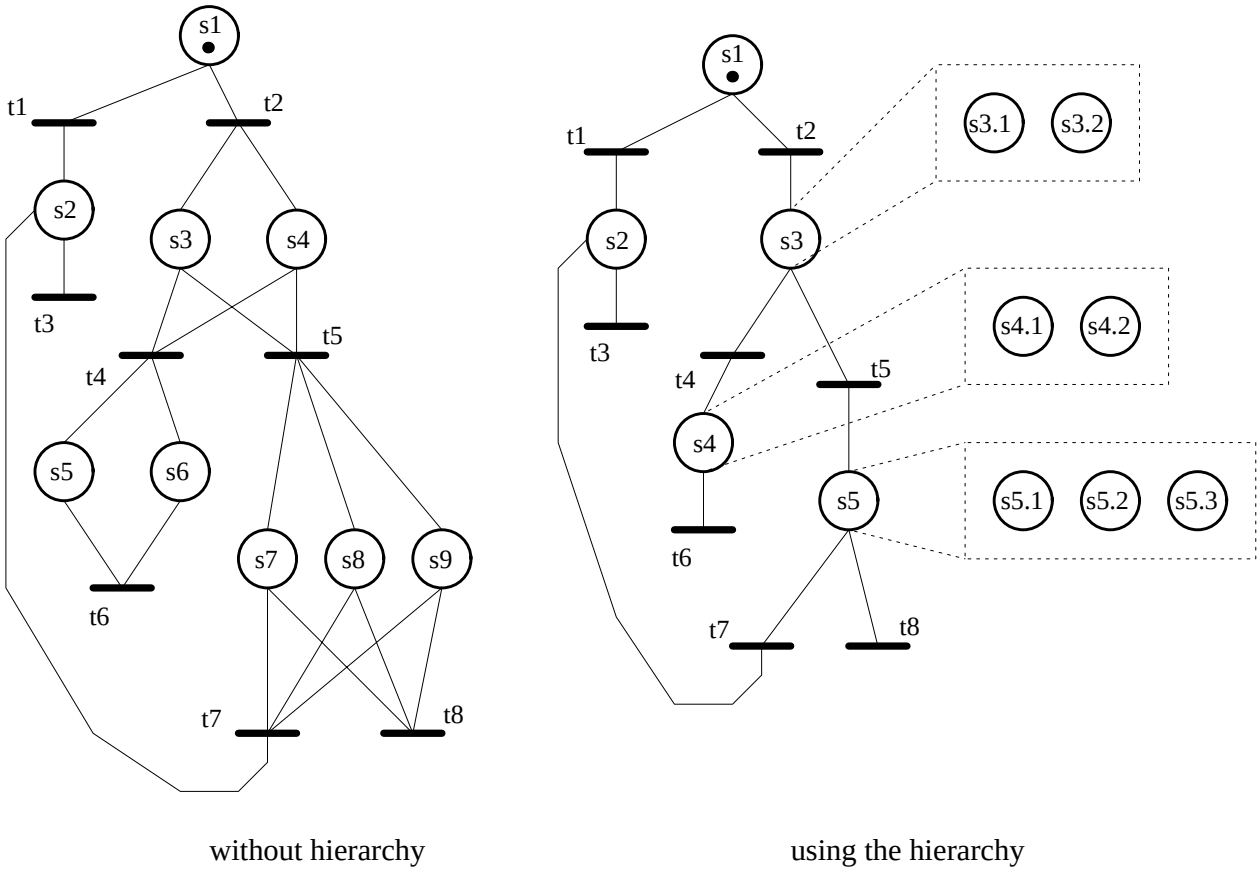


Figure 7: Petri net representation of guided tours.

at each node, we view it as splitting one browsing session into several distinct sessions (or, equivalently, several logically distinct readers). The discussion of colored tokens in section 5 provides a more detailed exposition of multiple readers in a hypertext.

It should also be noted that the Trellis model of hypertext neither prescribes nor prohibits any particular window placement or replacement strategy for concurrent displays. The particular choices of a hypertext system designer are reflected in the display mapping specified, but no form is inherently disallowed.

4.2 Concurrent browsing paths and synchronization

As a computation model, Petri net theory is recognized as an excellent representation of parallel activities. As such, it is a natural choice for representing concurrent display of multiple elements and concurrent browsing paths in a hypertext. The advantage obtained over directed graphs alone is that the execution semantics of Petri nets provide an author with the opportunity to specify synchronization of concurrent activity.

There are several forms in which concurrent activity may appear. First is simply the display of several (presumably related) content elements simultaneously in different windows or via different media. For example, selection of a single button may causes a browser to see a text frame, a related picture, and perhaps hear some audio as well. These distinct elements are, in essence, a unit and a single action should cause all three elements to be removed from the display when the reader is finished with them. This type of concurrent activity is created using a Petri net form in which a single transition (the button) branches to several places (the multiple elements) and a second transition is the sole output of these places. An example can be seen in the portion of Figure 2 composed of the transitions t_1 and t_2 along with places s_3, s_4, and s_5. The three content elements associated with these places will all be visible after button $B_l(t_1)$ is selected. They all then will be removed when either button $B_l(t_2)$ or button $B_l(t_{10})$ is selected.

As another example, consider the guided tours recently described by Trigg for Xerox's NoteCards [28].

A guided tour is formed from a pool of possible display windows ("cards"). The various cards of interest to an author are collected into sets called "tabletops." The cards in a tabletop are all displayed concurrently. An author constructs a tour by linking tabletops together to form a directed graph. From any one tabletop there may be several alternative next tabletops, and the final linked structure may be cyclic as well.

Guided tours are a subclass of the hypertexts one can describe with the Petri nets. The browsing control they offer can be represented by the Trellis model in two ways. The first method does not use hierarchy; rather, we assume that each place in a Petri net has an information element associated with it. A tabletop is then represented by a Petri net fragment consisting of a single transition connected to as many places as there are elements in the tabletop (with the exception of the first tabletop, which has no transition). Tabletop net fragments are interconnected as desired by the author. When one tabletop is connected to another, an arc is created from each of the places in the source fragment to the single transition in the destination fragment. This construction is illustrated by the left side of Figure 7. Note that lone transitions may follow terminating tabletops (for instance, transition t_6 follows places s_5 and s_6) to clear their elements from view at the end of a tour. To begin browsing, a token is placed in each of the places constituting the first tabletop of the tour.

The second construction technique uses the model's hierarchy, and it is more illustrative of the underlying sequential finite automaton in guided tours. The set of elements in a tabletop is fixed, so we represent each tabletop as a simple lower-level Petri net: one place for every information element in the tabletop, with no connected transitions. Then, at the higher level, a single place followed by a single transition will represent each tabletop (again with the exception of the first tabletop). A tabletop is linked by an author to others as in the previous method. The right side of Figure 7 shows this construction. The semantics for browsing a net hierarchy specify that a token arriving at a high-level tabletop place will cause all the places in the low-level net to be marked, and their contents thereby to be displayed. Since the lower level net representing a tabletop has no transitions, no browsing can occur at the lower level. The only button selections a reader can make correspond to the high-level level transitions that leave the high-level place representing the entire tabletop. For example, when browsing the tabletop represented by place s_3, content elements for places $s_{3.1}$ and $s_{3.2}$ will be visible, and buttons for transitions t_4 and t_5 will be selectable.

Since the high-level Petri net formed by the hierarchical construction method has a single input and a single output arc for each transition (the terminating transitions t_3, t_6, and t_8 can be thought of as all leading to a single "final state" place with null contents), it is equivalent to a deterministic finite state machine [23, page 41, ff.]. Thus, separate parallel browsing paths in a guided tour must be simulated unnaturally. Careful selection of the elements that appear in successive tabletops can give the appearance of parallel independent paths, but such paths cannot be identified from the linked structure alone.

Therefore, we introduce a second form of browsing concurrency—a generalization of multiple concurrent elements to multiple concurrent paths. Concurrent browsing paths are two or more sequences of content elements that are displayed at the same time. During browsing, concurrent paths may be synchronized and coalesced, they may join and overlap but still remain distinct, or they may never join again after their creation. The degree of relatedness among elements in two different paths can vary from complete independence to a strong dependence requiring rigid and frequent synchronization.

Separate concurrent browsing paths exist whenever multiple concurrent elements are created, and they remain distinct until they are synchronized and coalesced. When several parallel paths are active one content element from each is concurrently visible, but the speed of browsing along each path can vary according to the desires of the reader, within the limits prescribed by the structure. The author specifies the desired parallel path interactions based on an understanding of what net execution semantics will allow and what the content and context of the information requires. Thus a browser's experience with a Petri net based hypertext can be somewhat under the author's control.

Different parallel browsing effects are easily created by varying the placement and frequency of synchronizing events in a Petri net structure. Figure 2 contains some simple paths that are independent for a while but then are synchronized and coalesced. Firing transition t_4 causes two concurrent paths to be created, showing first elements $C_l(s_7)$ and $C_l(s_8)$. Instead of having both elements immediately removed thereafter, browsing continues along the rightmost path only. Firing transition t_5 causes the content element of place

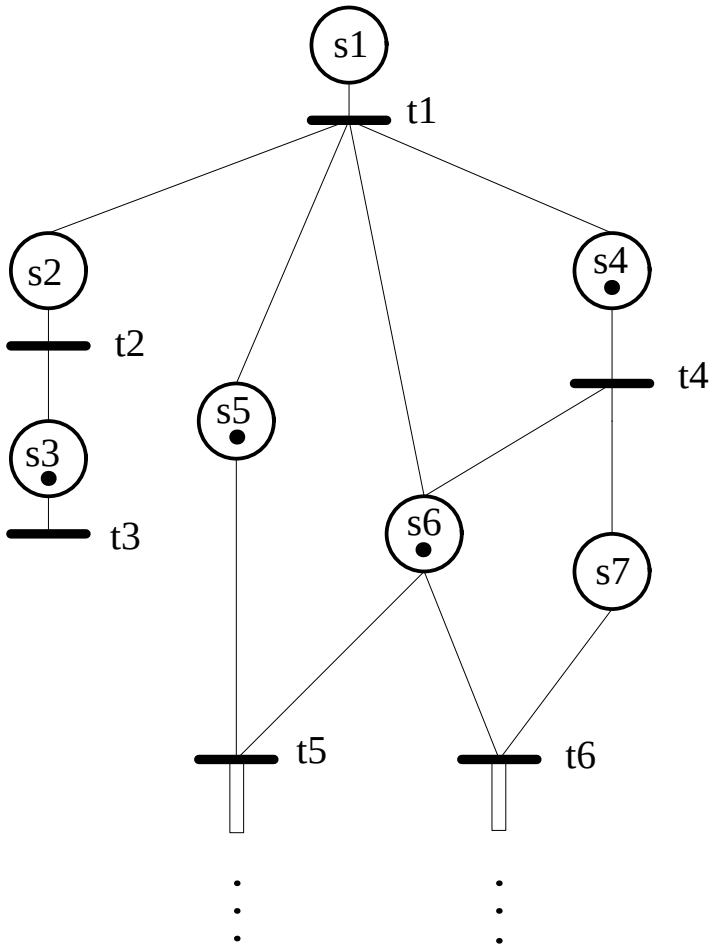


Figure 8: Multiple independent browsing paths.

s_9 to replace the element for place s_8, while the element for place s_7 remains visible. Transition t_6 is not enabled for firing (and, therefore, the button $B_l(t_6)$ is not visible for selecting) until after transition t_5 is fired. Firing t_6 then removes both content elements from view and coalesces the two browsing paths into one. This example is short, but the principle generalizes to arbitrarily long paths with numerous parallel branches. Even though browsing activities are concurrent on the different paths, they are not allowed to progress at speeds that are too dissimilar. In essence, synchronization points required the reader to "catch up" on each of the multiple paths before continuing on any of them.

A synchronized structure like this can be used for presenting information simultaneously at several levels of detail, or in several complementary forms. For example, using only the net structure in Figure 2, the content mapping could be one that maps place s_7 to, say, a picture of the NASA space shuttle. The content elements associated with places s_8 and s_9 could then be textual descriptions of some aspect of the picture, say an explanation of the orbiter vehicle itself and a description of the booster rockets respectively.[5] The Petri net structure shown dictates that the text descriptions are to be displayed in sequence, and the picture of the shuttle is to remain visible during the entire sequence. When the text frame sequence is consumed, selecting the single button $B_l(t_6)$ would remove all the related concurrent windows from view. The burden of cleaning up multiple displays can thus be removed from the reader by the synchronization inherent in a Petri net automaton.

Concurrent browsing paths do not have to be synchronized or coalesced. They can overlap like a pipeline,

[5] This example was suggested to us by a novel graphic display of the NASA space telescope in an experimental Sun workstation version of Shneiderman's Hyperties system [25, 21].

join and separate, or just individually come to an end. Figure 8 shows a Petri net structure in which four parallel browsing paths are active. From an initial marking $M_0 = (1000000...)$, transition t_1 is fired to create the next marking $M_1 = (0101110...)$. Transition t_2 is then fired to create the marking $M_2 = (0011110...)$ shown in the figure. The contents of the four places s_3, s_4, s_5, and s_6 are all concurrently visible. From this point the reader can select three different buttons. Selection of button $B_l(t_3)$ will cause the leftmost path to simply terminate, removing the contents of s_3 and replacing it with nothing. Selection of button $B_l(t_5)$ instead will synchronize the middle two paths and move them further along in the net. Selection of button $B_l(t_4)$ though is perhaps the most interesting of these cases. This action will create a net marking of $(0010121...)$, putting a second token in place s_6. The rightmost path continues to exist separately from the middle two; they are not coalesced into a single path as can happen at a synchronization point. The paths simply come together and overlap here. Even with two tokens present in s_6, its content element is displayed in only one spot. When either of transitions t_5 or t_6 is fired, though, a token will remain in s_6 so its contents will remain displayed. One path will move on, and another will remain at s_6.

Note that in this example, buttons can come and go within the context of one content element. When $C_l(s_6)$ is displayed, for instance, there may sometimes be only one button shown (for t_5), as in the scenario of Figure 8. At other times two buttons may be shown for this same element. For instance, from marking $M_1 = (0101110...)$ transition t_4 can be fired to create a next marking $M_2 = (0100111...)$. In this state, both transitions t_5 and t_6 are enabled so their corresponding buttons will both be displayed with the contents of place s_6. In this particular marking, the button for t_6 will also appear in another window with the content element for place s_6.[6]

An example of a hypertext that might effectively use multiple browsing paths is one in which the reader needs to accomplish several independent tasks, say locating a record in a personnel file and locating a record in a payroll account. The relative speeds with which the browsing tasks are accomplished are unimportant, since they are essentially independent.

A variant of the multiple path form of concurrency is obtained using a Petri net extension called *colored tokens* [19]. Here, each token is associated with one of several classes, and firing a transition requires all input places to be marked with like-colored tokens. Using this model, completely independent browsing sessions can be maintained. For example, a single hypertext can be shared and browsed by different readers without them interfering with each other. Each reader is given a different token color. More discussion on the use of colored Petri nets in hypertext can be found in Section 5.

4.3 Node reachability and unreachability

As with directed graphs alone, the Petri net model can be used to determine if portions of a hypertext can actually be reached during browsing, or alternately, if portions can *never* be reached. This latter consideration forms the basis for a unified method of providing hypertext access control, as discussed in the next section. Given a hypertext H, an initial marking M_0, and a place s, to determine if a particular content element $C_l(s)$ can be viewed during browsing, one must simply compute the reachability graph for the marked hypertext $< H,\ M_0 >$ and then scan the nodes looking for a state in which place s is marked. If no state exhibits such a marking, then the information cannot be viewed when the hypertext is browsed starting in state M_0. Similarly, we can determine if certain collections of information can be viewed simultaneously by looking for states containing sets of marked places.

Another characteristic of a hypertext that can be determined from the reachability graph is termination. If a state M exists in which no transitions are enabled, then there can be no next state M' from M, so the browsing session in that Petri net must terminate in M. Such terminal states appear in the tree representation of reachability graph as leaves that are not duplicates of other nodes. If the author desires that a hypertext have no terminal states, then this property can be easily checked by scanning all nodes in the graph.

[6] This non-static link behavior is an important feature of van Dam and Feiner's Document Presentation System [15], which is further described by van Dam [30, page 894].

Similarly, an author may wish to verify that any terminal states in a browsing session are ones in which the screen is blank, i.e., all places are unmarked in a terminal state. Another state property that is reasonable to look for is that a browsing session can return to its initial state, thus making it cyclic. Any characterization one wishes to describe for a marking can be checked against the reachability graph in order to verify that a hypertext has or fails to have a particular property.

Another class of properties that can be checked for a browsing session are temporal relationships among visitations of content elements. A Petri net, being an automaton, can be viewed as a language generator. The language of a Petri net [23, page 151, ff.] is the set of all sequences of transitions that can be fired during execution. In hypertext terms, the corresponding concept is the button sequences a reader might select during browsing from an initial marking. Consequently, analysis of the language of the Petri net can be used to verify the browsing requirements an author may have for a hypertext. For example, an author may wish to ensure that any browsing session that encounters an element Y must have somewhere previously encountered element X (e.g., X is a legally-required statement that protects trade secrets divulged in Y). Another example is ensuring that any browsing session encountering an element X *cannot* subsequently encounter element Y, but a browsing session that first encounters Y *can* later encounter X (e.g., X is an expert-level explanation of a topic and Y is a novice-level explanation of the same topic). These are difficult properties to verify by inspection in a complicated net.

4.4 Access control

The Petri net formalism can be effectively used to enforce browsing restrictions on readers of a hypertext. In a hypertext represented as a plain (unannotated) directed graph, if two browsing paths share a common node, then no reader can be prevented from visiting other nodes that lie on paths leaving that common node. With an unannotated Petri net, however, browsing paths can shared numerous common subsections and still have interspersed mutually exclusive sections as well. The limitation of plain directed graphs is usually dealt with in hypertext systems by annotating arcs with keywords or attributes, and controlling arc access via predicates. The Neptune system [11], for instance, uses a list of attribute/value pairs on nodes and arcs to provide conditional browsing of hypertext sections. Attributes in Neptune can be dynamically created and deleted by readers, a feature that the Trellis model does not include because of a stronger distinction between a hypertext's author and its readers.

The access control capabilities described in this section are otherwise not different from those available with graph augmentation techniques such as tagged arcs. However, we believe the Petri net notation is as intuitively easy to use as existing augmentation techniques, if not easier, while being more succinct and convenient. The notation successfully describes different existing access control mechanisms in one unified mathematical formalism, making them accessible to comparative reasoning and analysis.

With techniques like tagged arcs, the manner in which the graph additions are used by a browser is not inherently obvious. The exact semantics of their behavior and capabilities depend on the hypertext implementation and how it employs the added notation. With the Petri net model, access capabilities become an integral part of a document as opposed to being properties of the browser. What this really implies is that a Petri net engine essentially provides a standardized abstract browser implementation, one whose semantics are mathematically defined as opposed to being "code defined."[7] The unified execution abstraction of access control integrated with document structure is an important difference from augmentation techniques like tagged arcs and predicates.

Access control in the Trellis model uses the idea of a marked hypertext. For a hypertext H, various classes U_i of users can be identified, depending on which portions of H the author desires to be visible to certain readers. Then each class can be given a different starting state M_0^i for browsing H. Thus, each class U_i constitutes a separate marked hypertext $H_{M_0^i} =< H,\ M_0^i >$.

For example, consider a scenario in which an employment record will have two classes of reader: *privileged* (class P) and *restricted* (class R). Readers of class P can browse the entire document, but readers of class R

[7] The Hypertext Abstract Machine [4], the basis of the Neptune system, is a similar attempt to provide an abstract system semantics for standardization of browsing behavior.

may not see the sections containing, for example, job performance evaluations. We refer again to the Petri net structure in Figure 2 to represent this scenario. Let places s_3 through s_5 have job evaluations as their images under mapping C_l; as discussed earlier, the structure specifies that the three are to be displayed concurrently. Places s_{12} and s_{13} also contain privileged information, and their contents are to be displayed simultaneously after the evaluations have been read. The rest of the employment record (s_2, s_6 through s_9, and s_{11}) is unrestricted information. Places s_1 and s_{10} are special "access control" places that serve to separate the classes of reader during browsing. On this document, readers of class P define a hypertext with initial Petri net marking $M_0^P = (1100000001000)$, whereas readers of class R define a hypertext with initial marking $M_0^R = (0100000000000)$. The initial marking shown in Figure 2 is that of class P. The reachability graph resulting from this marking is shown in Figure 6(a). Having a token in each of places s_1 and s_2 enables transition t_1. This in turn allows readers of class P to select the button $B_l(t_1)$ and thereby read the job performance evaluations. Readers of class R do not start with a token in the access control place s_1, and a scan of the reachability graph for their initial hypertext state, shown in Figure 6(b), shows that no token can ever be there. Since transition t_1 is never enabled for readers of class R, the evaluations can never be browsed.

Readers of class P can also choose to leave the restricted portion of the hypertext by selecting the button $B_l(t_{10})$ after reading the evaluations at places s_3 through s_5. The token in access control place s_{10} allows them to re-enter the restricted section at a later point. Firing transition t_7 replaces this access control token. Class R readers will never have a token in place s_{10}, so they will always be excluded from sections s_{12} and s_{13} of the hypertext (again see Figure 6(b)). Note that for both access control places in this example, the net structure is such that firing the transitions they guard replaces their respective tokens.

A third class of reader can be defined on this hypertext: *semi-restricted* (class S). Readers of class S can see the salary information contained in places s_{12} and s_{13}, but they cannot see the performance evaluations in places s_3 through s_5. This restriction can be created by giving readers of class S an initial marking $M_0^S = (0100000001000)$. An examination of the reachability graph for this marked hypertext shows that the token in access control place s_{10} will allow a reader to fire transition t_7, but that transition t_1 can never be enabled.

The structure of this simple example suggests the following strategy for constructing hypertexts with access control classes. First, outline the access control classes for the hypertext. Next, construct a collection of sub-hypertexts, each having a separate Petri net structure and each intended to be read *in its entirety* by access control class members. Next, for each sub-hypertext, decide which of the classes can browse it. Finally, using the hierarchy in the Trellis model, construct a top-level hypertext that has one place representing each sub-hypertext and that is connected with the desired browsing patterns. Add one access control place for each sub-hypertext place, connected like the ones in Figure 2 as input and output to the transitions before and after the sub-hypertext place. For each access control class, prescribe an initial marking that includes a token in the access control place of every sub-hypertext the class is allowed to browse.

4.5 Tailored versions of a hypertext

Tailored versions of a hypertext are useful when each is to contain information specific to a separate environment. Portions of the contents of the documents in a collection of such tailored versions are common to all of the versions and other portions are specific to one or to a subset of the documents. An example of a situation in which a tailored document set would be useful is to describe the invocation and use of a piece of software that runs on a number of different computers or operating systems. A similar situation is to support different user groups with different kinds of expertise (for example, a spreadsheet might be described for a user community that was expert in computer use or alternately for a group that was expert in accounting principles).

While each version could be maintained as a separate hypertext, a more attractive design is to take advantage of the commonality of structure and content in the collection of tailored versions, and to represent the collection as a single Petri net. The issue of representation and of specification of appropriate alternate choices for content is exactly the same issue discussed in the previous section on access control in browsing,

and the same solutions apply. The reachability graph analyses needed to verify the correctness of tailored versions may differ, however, from those defined for browsing control. One property to be shown for tailored versions is that *exactly one* of a set of places is reachable (i.e., the content element for only one of the versions is ever viewable). A second property to be shown is that every place in a particular set is reachable (i.e., all the contents of one version are potentially viewable).

It is interesting to note that tailored versions of a hypertext may also be specified through modification of the function C_l. The mapping defined by C_l would be the same for all common portions of the two versions, but would differ for the changed content elements. For example, an author may wish to create two parallel versions of a document, one written in English and the other in French. In this case, two different mappings would be associated with the Petri net places: one to the English-language contents, the other to the French-language contents. In this case, there would be no overlap between the content elements in the two versions. Similar substitutions may be used to tailor versions of a document for different environments—for example a user manual describing software that runs on a number of different pieces of hardware, or perhaps a descriptive document with versions at different levels of technical complexity corresponding to the technical knowledge of differing communities of readers. In these cases, only a subset of the content elements would vary among different versions.

Similar effects have been studied in the context of traditional documents using attribute-value pairs as the means for identifying the differing alternatives (for example, see Ilson's recent paper [18]). Within the context of the Trellis model, it is possible to verify that two versions based on the same logical structure contain corresponding collections of elements and exhibit the same connectivity. Browsers based on the Trellis model can easily be designed to permit simultaneous viewing of the separate versions, and to provide additional tools to help an author ensure that two versions are consistent with each other.

5 Discussion and implications of the model

Consider again the common representation of a hypertext as a (usually annotated) directed graph in which a single element is visible at any time during browsing. The Trellis model does not restrict this representation; rather, it generalizes it with concurrency and synchronization. This claim is supported by the observation that any directed graph can be expressed as a Petri net by a well-known simple transformation [23, page 41, ff.]. The transformation takes the directed graph that represents such a hypertext, makes each existing node a Petri net place, and creates a Petri net transition in the middle of each arc. Nets constructed by this technique are in an equivalence class with finite state machines. The browsing semantics under our model for the hypertexts are exactly as expected for the original directed graph form. Thus, methods applicable to directed graphs, such as arc annotations and history mechanisms, are equally applicable to Petri net based hypertexts.

In addition to directly representing graph based documents in Petri net form, a system based on the Trellis model can also adopt many of the user interface techniques that have been developed for browsing directed graph hypertexts. The Petri net provides the same logical document structure that a directed graph does in a browsing tool. The results of empirical studies into visual aspects of a hypertext browsing system (for example, see Shneiderman's paper [25]) are relevant when applied to either representation. However, research is needed into effective window placement strategies for multiple concurrent elements.

Several points about Petri net representation deserve further emphasis. First, the Trellis model provides the author of a hypertext with greater control over the sequences in which nodes will be browsed. Other researchers have found a similar need to permit this kind of specification (both within one document and among multiple documents [6, 7, 32, 33]). However, the Petri net model permits flexible enforcement of such sequencing, making browsing restrictions an integral part of a hypertext's structure rather than applying it with an external browsing mechanism. As such, we believe the Trellis model provides a needed unified structuring facility to the author. Furthermore, as with other browsing control facilities, the control provided by Petri nets is not forced on an author. The fact that unrestricted directed graphs (i.e., finite state machines) are a subclass of Petri nets implies that an author can choose to apply no browsing control if that is the effect

desired. Concurrency and sequencing control are available, though, and can be specified to whatever degree an author requires. Finally, augmentation techniques that are applicable to directed graph hypertexts are equally applicable to Petri nets. Essentially, a hypertext system based on highly-annotated directed graphs can have its document representation replaced by Petri nets, thereby gaining concurrency specification and control without sacrificing any existing functionality.

Two other advantages of the Trellis hypertext model should be noted. First, the Petri net representation is essentially graphical and consequently is only an incremental change from the common directed graph model of hypertext.[8] Secondly, Petri nets have been studied and analyzed for over twenty years, so an extensive theory exists that can be immediately applied to solution of problems in the hypertext domain. Indeed, useful extensions have been defined to Petri nets that will have direct application to hypertext as well. Two such extensions are deterministically timed Petri nets and colored Petri nets.

Use of a deterministically timed Petri net [9, 27] in the model allows hypertext documents such as timed exams and paced presentations, or even simple animations. Several types of timed Petri nets have been well developed, and a body of results are available for determining such temporal properties as relative firing frequencies of transitions, or execution durations of subnets.[9] In the timed Petri net model that seems most appropriate for hypertext, an integer is associated with each place in the Petri net; it represents the number of time units (clock "ticks") that must pass before a token in the place can enable any of its output transitions to fire. In hypertext terms, the integer time on a place indicates the minimum amount of time that the contents of a place are displayed when a token arrives at the place. After the requisite number of ticks have passed, the token would participate in enabling transitions for firing, meaning either that selectable buttons would appear in display windows, or, if timing were to be strictly enforced, transitions would be fired automatically, thereby causing the displays to change.[10]

Colored Petri nets [19] extend the classical model by associating colors with the tokens, the places, or perhaps the transitions of a net. A color is simply a method of distinguishing classes of elements that share the same structural category. The firing rule is altered to require some color property to hold in addition to having tokens in all input places for an enabled transition. For example, the rule may require all tokens being consumed by a transition to be of the same color, or to be all of the same color *and* to be the color of the transition. Color extensions increase the size of the state space that can be represented with a given net structure.

Several applications of colored Petri nets are of note in the hypertext domain. In one obvious extension, colored tokens permit multiple simultaneously-active asynchronous browsing sessions over a hypertext. Additional sessions cannot be created by simply adding tokens to an uncolored net, as this could permit traversal of otherwise prohibited paths. In a colored net, a new independent browsing session would be generated by replicating the initial marking for the net (or perhaps the current marking) with a previously-unused token color. This preserves the correctness of the author-specified net behavior, as tokens of differing colors do not interact. The issues surrounding the actual mapping of separate browsing sessions to a display, or a network of displays, are outside of the scope of this paper, and indeed schemes such as Lifshitz and Shneiderman's [20, 21] could be adopted to provide this partitioning.

Another application for colored Petri nets is to permit definition of additional levels of granularity in access control. KMS [1, page 832] has considered the case of hypertext nodes that are both readable and modifiable by browsers (for instance, to permit addition of notations and commentary). At issue is how to grant some browsers the right to modify while restricting others to reading only. One approach might be

[8] Contrast the use of this graphically-based model with a non-graphical one such as Garg's, which is based in abstract algebra [16]. While the non-graphical models have certainly demonstrated their utility, the mental shift required to employ their results is significant.

[9] See, for example, the proceedings of the 1985 International Workshop on Timed Petri Nets and the proceedings of the 1987 International Workshop on Petri Nets and Performance Models.

[10] Dami, et al. [10], describe a general timed computation mechanism called *temporal scripts* that has similar goals in that animated graphical presentations are principle examples in their paper. The approach in temporal scripts is to allow an object to consume virtual time as a resource, and have time slices passed around from agent to agent to effect computation. In lieu of timing on a Petri net, this model appears to be one that can also be used in conjunction with Trellis to specify and enforce more precise temporal properties for events during hypertext browsing.

to encode the access rights in the token's color, and to generalize the logical projection P_l and the display projection P_d to associate an appropriate presentation with each color. Browsers receiving the appropriate color of marked hypertext would then be granted modification privileges as they browse.[11]

6 Conclusions

In summary, we have presented a Petri net model of hypertext that generalizes the directed graph models currently in use. Since a Petri net is a directed graph, the Trellis model specifies the logical linked structure of a hypertext; being automata as well, Petri nets provide standardized browsing semantics for hypertext systems. As a natural formalism for concurrency, Petri net notation also provides a unique mechanism for the expression and control of concurrent browsing paths in a hypertext. This browsing control can be applied to or excluded from a document at the author's discretion. Since the finite state machine representation used in many hypertext systems is given by a subclass of Petri nets, useful annotations like tagged arcs and predicates can be as easily applied to the Trellis model as they can to directed graphs. The Trellis model, then, provides a unifying formalism for describing and reasoning about browsing behavior, both concurrent and sequential, both controlled and unrestricted.

An important research problem now is the development of composition techniques to produce "well formed" documents, and the incorporation of these techniques into a comprehensive authoring and analysis environment. One example of such an approach is the heuristic mentioned earlier for constructing a hypertext with access control classes. Another approach is an authoring language that we envision to structure meaningful hypertexts by the repeated composition of a few simple Petri net fragments, much as control structures in modern programming languages alleviate the confusion of an unstructured assembly code program [26].[12] The authoring language not only ensures that properly structured documents are produced, but the procedural programming paradigm hierarchically organizes a hypertext and helps manage the complexity of constructing and analyzing a very large document. An authoring environment based on the Trellis model and this structured programming approach to document construction must provide tools for writing in the language, as well as analysis tools for verifying that a hypertext will behave as expected.

The merging of hypertext and Petri nets is proving to be an advantageous combination. Much of the power of the model and corresponding implementation results from our ability to adopt already-established user interface techniques from the hypertext community and analytical techniques from the Petri net community. We believe that this formal model will allow the development of hypertexts that are not only general but also predictable in their behavior.

Acknowledgements

The authors gratefully acknowledge Frank Halasz, Robert Allen, and the anonymous referees for their thorough reading of earlier versions of this report. Their detailed comments and references were especially helpful in presenting this work more clearly and accurately.

References

[1] Akscyn, R. M., McCracken, D. L., and Yoder, E. A. KMS: A distributed hypermedia system for managing knowledge in organizations. *Commun. ACM 31*, 7 (July 1988), 820–835.

[11] We note that this use of colored tokens is a special case of capability based addressing as defined in the HYDRA operating system [31]. The more general statement would be that we associate a capability with the token and that the actions carried out in the P_l and P_d mappings would be defined based on that capability.

[12] Van Dam makes a similar point in his call for identification of new flow of control kinds of hypertext constructs [30, page 894].

[2] Brown, P. J. Hypertext: The way forward. In *Document Manipulation and Typography*, J. C. van Vliet, Ed. Cambridge University Press, Apr. 1988, pp. 183–191. Proceedings of the International Conference on Electronic Publishing, Document Manipulation, and Typography, Nice (France), April 20–22, 1988.

[3] Bush, V. As we may think. *The Atlantic Monthly 176*, 1 (July 1945), 101–108.

[4] Campbell, B., and Goodman, J. M. HAM: A general purpose hypertext abstract machine. *Commun. ACM 31*, 7 (July 1988), 856–861.

[5] Carmody, S., Gross, W., Nelson, T. E., Rice, D., and van Dam, A. A hypertext editing system for the /360. Tech. rep., Brown University, Center for Computer and Information Sciences, Providence, R.I., Mar. 1969. Also contained in M. Faiman and J. Nievergelt, editors. *Pertinent Concepts in Computer Graphics*. University of Illinois, Urbana, Ill., March 1969, pp. 291–330.

[6] Christodoulakis, S., Ho, F., and Theodoridou, M. The multimedia object presentation manager of MINOS: A symmetric approach. In *Proceedings of ACM SIGMOD '86* (May 1986), pp. 295–310. Washington, DC.

[7] Christodoulakis, S., Theodoridou, M., Ho, F., and Papa, M. Multimedia document presentation, information extraction, and document formation in MINOS: A model and a system. *ACM Trans. Office Inf. Syst. 4*, 4 (Oct. 1986), 345–383.

[8] Conklin, J. Hypertext: An introduction and survey. *Computer 20*, 9 (Sept. 1987), 17–41.

[9] Coolahan, J. E., and Roussopoulos, N. A timed Petri net methodology for specifying real-time system timing requirements. In *Proceedings of the International Workshop on Timed Petri Nets* (July 1985), pp. 24–31. Torino, Italy.

[10] Dami, L., Fiume, E., Nierstrasz, O., and Tsichritzis, D. Temporal scripts for objects. In *Active Object Environments (Environnements d'Objets Actifs)*, D. Tsichritzis, Ed. Centre Universitaire D'Informatique, Universite de Geneve, June 1988, pp. 144–161.

[11] Delisle, N., and Schwartz, M. Neptune: A hypertext system for CAD applications. In *Proceedings of ACM SIGMOD '86* (May 1986), pp. 132–143. Washington, DC.

[12] Delisle, N. M., and Schwartz, M. D. Contexts—A partitioning concept for hypertext. *ACM Trans. Office Inf. Syst. 5*, 2 (Apr. 1987), 168–186.

[13] Engelbart, D. C., and English, W. K. A research center for augmenting human intellect. *Proceedings, AFIPS Fall Joint Computer Conference 33* (1968), 395–410.

[14] Engelbart, D. C., Watson, R. W., and Norton, J. C. The augmented knowledge workshop. ARC Journal Accession Number 14724, Stanford Research Center, Menlo Park, Calif., Mar. 1973. Paper presented at the National Computer Conference, June 1973.

[15] Feiner, S., Nagy, S., and van Dam, A. An experimental system for creating and presenting interactive graphical documents. *ACM Trans. Gr. 1*, 1 (Jan. 1982), 59–77.

[16] Garg, P. K. Abstraction mechanisms in hypertext. *Commun. ACM 31*, 7 (July 1988), 862–870, 879.

[17] Halasz, F. G. Reflections on NoteCards: Seven issues for the next generation of hypermedia systems. *Commun. ACM 31*, 7 (July 1988), 836–852.

[18] Ilson, R. Interactive effectivity control: Design and applications. In *Proceedings of ACM Conference on Document Processing Systems* (December 5–9, 1988, Santa Fe, New Mexico) (Dec. 1988), ACM, New York, pp. 85–91.

[19] Jensen, K. Coloured Petri nets and the invariant method. *Theoretical Comput. Sci. 14* (1981), 317–336.

[20] Lifshitz, K., and Shneiderman, B. Window control strategies for on-line text traversal, July 1987. Working paper.

[21] Marchionini, G., and Shneiderman, B. Finding facts vs. browsing knowledge in hypertext systems. *Computer 21*, 1 (Jan. 1988), 70–80.

[22] Molloy, M. K. A CAD tool for stochastic Petri nets. In *Proceedings of the ACM-IEEE Fall Joint Computer Conference* (Nov. 1986), pp. 1082–1091.

[23] Peterson, J. L. *Petri Net Theory and the Modeling of Systems.* Prentice-Hall, Inc., 1981.

[24] Reisig, W. *Petri Nets: An Introduction.* Springer-Verlag, 1985.

[25] Shneiderman, B. User interface design for the Hyperties electronic encyclopedia. In *Proceedings of Hypertext '87* (Nov. 1987), pp. 189–194. Published by the Association for Computing Machinery, 1989.

[26] Stotts, P. D., and Furuta, R. Alphα: An authoring language for Petri-net-based hypertext, 1989. Hypertext 2, University of York, June 29th and 30th, 1989.

[27] Stotts, Jr., P. D., and Pratt, T. W. Hierarchical modeling of software systems with timed Petri nets. In *Proceedings of the International Workshop on Timed Petri Nets* (July 1985), pp. 32–39. Torino, Italy.

[28] Trigg, R. H. Guided tours and tabletops: Tools for communicating in a hypertext environment. In *Proceedings of Conference on Computer-Supported Cooperative Work* (September 26–29, 1988, Portland, Oregon) (1988), pp. 216–226.

[29] van Biljon, W. R. Extending Petri nets for specifying man-machine dialogues. *International Journal of Man-Machine Studies 28* (1988), 437–455.

[30] van Dam, A. Hypertext '87 keynote address. *Commun. ACM 31*, 7 (July 1988), 887–895.

[31] Wulf, W., Cohen, E., Corwin, W., Jones, A., Levin, R., Pierson, C., and Pollack, F. HYDRA: The kernel of a multiprocessor operating system. *Commun. ACM 17*, 6 (June 1974), 337–345.

[32] Zellweger, P. T. Directed paths through collections of multi-media documents. In *Hypertext '87* (Nov. 1987). Position paper.

[33] Zellweger, P. T. Active paths through multimedia documents. In *Document Manipulation and Typography*, J. C. van Vliet, Ed. Cambridge University Press, Apr. 1988, pp. 19–34. Proceedings of the International Conference on Electronic Publishing, Document Manipulation, and Typography, Nice (France), April 20–22, 1988.

[34] Zisman, M. D. Use of production systems for modeling asynchronous, concurrent processes. In *Pattern-Directed Inference Systems*, D. A. Waterman and F. Hayes-Roth, Eds. Academic Press, Inc., 1978, pp. 53–68.